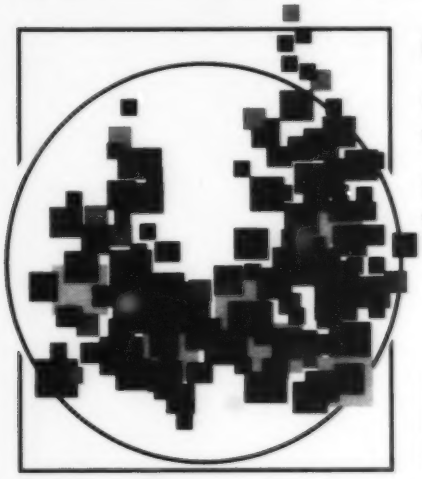

CENTRE FOR
SYSTEMS
SCIENCE

LCCR

Simon Fraser University, Burnaby, B.C., Canada V5A 1S6 / (604) 291-4704

Efficient Implementation of Cautious Schedulers

Naoki Katoh, Toshihide Ibaraki and Tiko Kameda

CSS/LCCR TR90-13

August 1990

92-05796/

Efficient Implementation of Cautious Schedulers†

Naoki Katoh‡
Tiko Kameda*
Toshihide Ibaraki§

Abstract

In this report, we discuss the implementation of three cautious schedulers *CS*(*WW*), *MCS*(*MWW*), and *MCS*(*MWRW*), which we introduced earlier. Our results show that the time complexities of these three schedulers are similar to the two-phase locking scheduler in which deadlock detection procedure is invoked every time a new request arrives.

---

† This work was supported in part by the Natural Sciences and Engineering Research Council of Canada and in part by the Ministry of Education, Science and Culture of Japan under Scientific Research Grant-in-Aid.

‡ Department of Management Science, Kobe University of Commerce, Kobe, 673 Japan.

* School of Computing Science, Simon Fraser University, British Columbia, Canada V5A 1S6. Netaddress: tiko@cs.sfu.ca

§ Department of Applied Mathematics and Physics, Kyoto University, Kyoto, 606 Japan. Netaddress: ibaraki@kuamp.kyoto-u.ac.jp

1. Introduction

Cautious schedulers have been extensively investigated in [6-8]. Similar schedulers are also called **type 1 schedulers** [2, 3] or **pessimistic schedulers** [1]. We assume the reader is familiar with the basic idea behind cautious scheduling. In this paper, we consider three cautious schedulers $CS(WW)$, $MCS(MWW)$, and $MCS(MWRW)$, where WW, MWW, and $MWRW$ are defined in [5].

Let P be the partial schedule of read and /or write steps that have already been granted by the scheduler, q be the current step being tested, and $PEND$ be the set of **pending** steps, i.e., those predeclared by transactions but not granted yet. The **active TIO graph** [6] (or **ATIO graph**, for short), denoted by $ATIO(Pq, PEND)$ (or $ATIO(<P, I>, q, PEND)$ in the multiversion case) has a node set that consists of the transactions whose steps are in Pq and/or $PEND$ and some **dummy nodes**, which are the terminal nodes of the dummy arcs defined below. Its arc set consists of two disjoint subsets, A and A′. Subset A contains the **read-from arcs** and **dummy arcs**. A read-from arc $(T_i, T_j):X \in$ A represents the fact that transaction T_j has read the value of data item X from T_i. A dummy arc $(T_i, T_i'):X \in$ A represents a write operation $W_i[X]$ such that no read operation in Pq (or P in the multiversion case) has read X from T_i. Each write operation $W_i[X]$ of a step in $\{q\} \cup PEND$ is represented by a dummy arc $(T_i, T_i'):X \in$ A′, called a **pending write arc**.

To take serialization constraints into account [5], we add constraint arcs to $ATIO(Pq, PEND)$ (or $ATIO(<P, I>, q, PEND)$ in the multiversion case). Let A and B be two steps in $Pq \cup PEND$.[1] We write $A < B$ if either A precedes B in Pq, or $A \in Pq$ and $B \in PEND$. For a given set c of constraints (e.g., c = ww, wrw [5]), if a step A of T_i and a step B of T_j such that $A < B$ give rise to a c-constraint (e.g., $A = W_i[X]$ and $B = W_j[X]$ if c = ww), then a constraint arc (T_i, T_j), called a ***c*-arc**, is introduced. For a set c of constraints, let $ATIO_c(Pq, PEND)$ (or $ATIO_c(<P, I>, q, PEND)$ in the multiversion case) stand for the active TIO graph augmented by the c-arcs. To be precise, more than just c-arcs are added to the ATIO

[1] We abuse the notation here and use Pq to denote the set of steps which the sequence Pq consists of. This should cause no confusion.

graph. For example, if $c = ww$, we add a wr-arc (T_k, T_h) for each pending read operation $R_h[X] \in PEND$, where $W_k[X]$ is the last write operation on X in Pq. These arcs are not really introduced as wr-arcs as such, since we are dealing with ww-constraints only. T_h will eventually read X from some T_i, where either $i = k$ or $W_i[X] \in PEND$, which implies that there will be a path from T_k to T_h. Similarly, if $c = mww$ [5], we add an rw-arc (T_j, T_h) for each pair of j and h, where $q = R_j[S]$ and $W_h[X] \in PEND$ with $X \in S$. (See rule (6) in Section 2.) The justification for these additional arcs is given in Section 2.

The concept of *disjoint-interval topological sort* (*DITS*, for short) was introduced in [5]. Since it will play an important role in this report, we repeat its definition here. A total order $\ll$ on the set of nodes of an ATIO graph is a DITS, if it satisfies the following two conditions:

(a) If $T_i \ll T_h$ then there is no path from T_h to T_i in the ATIO graph, and

(b) [*Exclusion Rule*] If $T_i \ll T_l$ then $T_j \ll T_h$, where $(T_i, T_j):X$ and $(T_h, T_l):X$ are any two arcs labeled by X in the ATIO graph such that $i \neq h$.

Let $(T_i, T_j):X$ and $(T_h, T_l):X$ be two arcs in $ATIO_c(Pq, PEND)$ (or $ATIO_c(<P, I>, q, PEND)$ in the multiversion case), where $i \neq h$. If there is a path in the graph from T_i to T_l (possibly through T_j or T_h), we introduce an unlabeled **exclusion arc** (T_j, T_h), unless a path already exists from T_j to T_h. Its **exclusion closure**, denoted by $ATIO^*_c(Pq, PEND)$ (or $ATIO^*_c(<P, I>, q, PEND)$ in the multiversion case), is obtained from $ATIO_c(Pq, PEND)$ (or $ATIO_c(<P, I>, q, PEND)$ in the multiversion case) by adding arcs (T_0, T_i) and (T_i, T_f) for all T_i $(\neq T_0, T_f)$, and all the exclusion arcs. Here $T_0 = W_0[\mathbf{D}]$ and $T_f = R_f[\mathbf{D}]$, where **D** is the set of all data items, are fictitious initial and final transactions introduced for serializability testing [5]. It can be shown that the exclusion closure is unique and does not depend on the order of adding the exclusion arcs.

It is shown in [6] and [7] that the current request q with a partial schedule P ($<P, I>$ for the multiversion case) and a pending set $PEND$ can be granted (i.e., Pq can be extended to a serializable schedule using the steps in $PEND$) by the above three schedulers, if and only if $ATIO^*_{ww}(Pq, PEND)$, $ATIO^*_{mww}(<P, I>, q, PEND)$, or $ATIO^*_{mrw}(<P, I>, q, PEND)$ is acyclic, respectively. Checking whether Pq can be extended to a serializable schedule or not is

called the **completion test**.

These ATIO graphs have many redundant arcs that result by transitivity from others. For $CS(WW)$ it is known [6] that instead of $ATIO^{*}_{ww}(Pq, PEND)$, the *ww-serialization graph* $SG_{ww}(Pq, PEND)$ can be used. This graph has fewer arcs and is acyclic if and only if the corresponding exclusion-closed ATIO graph is acyclic. Moreover, any topological sort of the serialization graph is a DITS for the corresponding exclusion-closed ATIO graph.

We review how $SG_{ww}(Pq, PEND)$ is constructed, and show that a similar approach can be used in the multiversion case. Namely, for $MCS(MWW)$ and $MCS(MWRW)$, we shall define the *mww-serialization graph* $SG_{mww}(<P, I>, q, PEND)$ and the *mwrw-serialization graph* $SG_{mwrw}(<P, I>, q, PEND)$, which have fewer arcs than the corresponding exclusion-closed ATIO graphs. We show that these serialization graphs are acyclic if and only if the corresponding exclusion-closed ATIO graphs are acyclic. Moreover, any topological sort of such a serialization graph is a DITS for the corresponding exclusion-closed ATIO graph.

There are other considerations for improving the time complexity of the completion test. First of all, it is wasteful to construct the required SG graph from scratch every time a new request arrives at the scheduler; it is better to maintain the SG graph and update it as needed. Secondly, since the number of nodes in the graph increases with time, we wish to delete those nodes which do not affect scheduling decisions. We shall discuss rules for such deletions.

With these time-saving measures, the time complexities of the above three schedulers are similar to that of the two-phase locking scheduler, if deadlock detection procedure is invoked every time a new request arrives. In view of the fact that these cautious schedulers allow more concurrency than the two-phase locking scheduler [9], they deserve a serious consideration in practical applications.

2. The Completion Test for $CS(WW)$ and $MCS(MWW)$

2.1 Review of the completion test for $CS(WW)$

Given a partial schedule Pq with a set $PEND$ of pending requests, the **completion test** for $CS(WW)$ is carried out as the acyclicity test of the exclusion-closed ATIO graph, $ATIO^{*}_{ww}(Pq, PEND)$, or equivalently, of the ww-serialization graph, $SG_{ww}(Pq, PEND)$ [6]. $SG_{ww}(Pq, PEND)$ is constructed from $ATIO(Pq, PEND)$ by first deleting all its dummy nodes and their incident arcs. Note that all the read-from arcs of $ATIO(Pq, PEND)$ are retained in $SG_{ww}(Pq, PEND)$. In particular, if $q = R_i[S]$ $(S \subseteq \mathbf{D})$, $SG_{ww}(Pq, PEND)$ has a read-from arc (T_k, T_i), where $W_k[X]$ is the last write operation on X in Pq, for each $X \in S$. Also, $SG_{ww}(Pq, PEND)$ has the following additional arcs for each $X \in \mathbf{D}$.

(1) a ww-arc (T_i, T_l) for each consecutive pair of write operations $W_i[X]$ and $W_l[X]$, appearing in this order in Pq,

(2) a ww-arc (T_k, T_h) for the last write operation, $W_k[X]$, on X in Pq and each pending write operation $W_h[X] \in PEND$,

(3) a wr-arc (T_k, T_h) for the last write operation, $W_k[X]$, on X in Pq and each pending read operation $R_h[X] \in PEND$,

(4) exclusion arcs (T_i, T_l) caused by each ww-arc (T_j, T_l) of type (1) above, where T_j $(j \neq l)$ read X from T_i, and

(5) exclusion arcs (T_j, T_h) caused by each ww-arc (T_k, T_h) of type (2) above, where T_j $(j \neq h)$ read X from T_k.

Remarks:

(a) All arcs of types (1), (2) and (3) above are also in $ATIO_{ww}(Pq, PEND)$.

(b) ww-arcs due to non-consecutive write operations are not drawn in $SG_{ww}(Pq, PEND)$, since they are implied by the ww-arcs of type (1).

Theorem 2.1: *$ATIO^{*}_{ww}(Pq, PEND)$ is acyclic if and only if $SG_{ww}(Pq, PEND)$ is acyclic.*

Proof. First note that all the read-from, constraint, and exclusion arcs introduced in $SG_{ww}(Pq, PEND)$ also exist in $ATIO^{*}_{ww}(Pq, PEND)$. Therefore, it is obvious that if $SG_{ww}(Pq, PEND)$ has a cycle, so does $ATIO^{*}_{ww}(Pq, PEND)$.

Conversely, assume that $SG_{ww}(Pq, PEND)$ is acyclic. It is sufficient to show that $ATIO_{ww}(Pq, PEND)$ has a DITS, since, as shown in [6], $ATIO_{ww}(Pq, PEND)$ has a DITS if and

only if $ATIO^{*}_{ww}(Pq, PEND)$ is acyclic. We first show that all the read-from and constraint arcs in $ATIO_{ww}(Pq, PEND)$ exist, or are implied by transitivity, also in $SG_{ww}(Pq, PEND)$. Clearly, each read-from arc in $ATIO_{ww}(Pq, PEND)$ exists as a read-from arc also in $SG_{ww}(Pq, PEND)$. Let us now consider the constraint arcs in $ATIO_{ww}(Pq, PEND)$. Let $W_i[X]$ and $W_j[X]$ be two write operations such that $W_i[X] < W_j[X]$. Then $ATIO_{ww}(Pq, PEND)$ has a ww-arc (T_i, T_j). We want to show that $SG_{ww}(Pq, PEND)$ has a path from T_i to T_j. If both $W_i[X]$ and $W_j[X]$ belong to Pq, then $SG_{ww}(Pq, PEND)$ has a path consisting only of ww-arcs introduced by rule (1). On the other hand, if $W_i[X]$ is in Pq and $W_j[X]$ belongs to $PEND$, then $SG_{ww}(Pq, PEND)$ has a path from T_i to T_k, where $W_k[X]$ is the last write operation on X in Pq, and it also has a ww-arc (T_k, T_j) introduced by rule (2). In addition to ww-arcs, $ATIO_{ww}(Pq, PEND)$ has wr-arcs that are exactly those introduced by rule (3).

Since $SG_{ww}(Pq, PEND)$ is acyclic, let σ be any one of its topological sorts. Construct σ' from σ by inserting each dummy node T_i' of $ATIO_{ww}(Pq, PEND)$ immediately after its "parent" T_i. We shall prove that σ' is a DITS of $ATIO_{ww}(Pq, PEND)$. Let us first note that every arc in $ATIO_{ww}(Pq, PEND)$ is directed from a transaction T_j to another transaction that follows T_j in σ'. This is obvious for dummy arcs, read-from arcs, and wr-arcs. For each ww-arc in $ATIO_{ww}(Pq, PEND)$, there corresponds a path consisting of one or more ww-arcs in $SG_{ww}(Pq, PEND)$, as discussed above.

To complete the proof we must show that every pair of edges $(T_i, T_j):X$ and $(T_h, T_l):X$ with $h \neq i$ satisfies the exclusion rule. If both of these arcs are dummy arcs, then this claim is clearly true, since each dummy node is located immediately after its parent in σ'. So let $W_i[X] < W_h[X]$ without loss of generality and assume that at least one of these two arcs is non-dummy.

Case 1: (T_i, T_j) is a non-dummy arc.

Subcase 1A: $W_h[X]$ belongs to Pq.

In this case, $W_i[X]$ occurs before $W_h[X]$ in Pq. Let $W_g[X]$ be the write operation on X such that $W_g[X]$ follows $W_i[X]$ in Pq without any intervening write on X ($g = h$ is possible). Then $SG_{ww}(Pq, PEND)$ has an exclusion arc (T_j, T_g) introduced by rule (4). If $g \neq h$, there is a path from T_g to T_h in $SG_{ww}(Pq, PEND)$ consisting of ww-arcs due to rule (1).

Therefore, $SG_{ww}(Pq, PEND)$ has a path from T_j to T_h. Thus two intervals (T_i, T_j) and (T_h, T_l) in σ' satisfy the exclusion rule.

Subcase 1B: $W_h[X]$ belongs to $PEND$.

If $W_i[X]$ is the last write operation on X in Pq, then $SG_{ww}(Pq, PEND)$ has an exclusion arc (T_j, T_h) by rule (5). If $W_i[X]$ is not the last write operation on X in Pq, on the other hand, let $W_k[X]$ be the last operation on X in Pq. Then it can be proved as in subcase 1A that $SG_{ww}(Pq, PEND)$ has a path from T_j to T_k. Since there is a ww-arc (T_k, T_h) by rule (2), there is a path from T_j to T_h in $SG_{ww}(Pq, PEND)$. This implies that two intervals (T_i, T_j) and (T_h, T_l) satisfy the exclusion rule.

Case 2: (T_i, T_j) is a dummy arc.

It is clear that $SG_{ww}(Pq, PEND)$, hence $ATIO_{ww}(Pq, PEND)$, has a path from T_i to T_h, consisting of a sequence of ww-arcs, which implies that (T_i, T_j) and (T_h, T_l) satisfy the exclusion rule. □

2.2 The completion test for *MCS*(*MWW*)

As $SG_{ww}(Pq, PEND)$ was obtained from $ATIO(Pq, PEND)$, $SG_{mww}(<P, I>, q, PEND)$ is obtained from $ATIO(<P, I>, q, PEND)$. However, there are three differences from the single version case, which reflect the difference between $ATIO_{ww}(Pq, PEND)$ and $ATIO_{mww}(<P, I>, q, PEND)$. First of all, arcs of type (3) are not included in $SG_{mww}(<P, I>, q, PEND)$, since old versions are available to be assigned to $R_i[X] \in PEND$. Secondly, if $q = R_j[S]$, the read-from arcs due to the current step are created *after* the acyclicity of $SG_{mww}(<P, I>, q, PEND)$ is verified by the completion test. If the acyclicity test succeeds, it means that there exists a set of **version assignments** to the current read request such that the $ATIO_{mww}(<P, I>, q, PEND)$ has a DITS when the read-from arcs reflecting the version assignments are incorporated in it. Thirdly, in addition to the arcs of types (1)-(2) and (4)-(5), the mww-serialization graph has

(6) an rw-arc from T_j to T_h for each pair of j and h, where $q = R_j[S]$ and $W_h[X] \in PEND$ with $X \in S$.

These arcs were introduced in $ATIO_{mww}(<P, I>, q, PEND)$; they are really exclusion arcs that will result from ww-arcs and the version assignments to $q = R_j[S]$. This prevents assigning $R_j[X]$ to a version to be written by a pending write on X. Note that they are introduced in $SG_{mww}(<P, I>, q, PEND)$ *before* the acyclicity test, and hence before the version assignments.

If $q = R_j[S]$, version assignment is carried out after finding a topological sort of $SG_{mww}(<P, I>, q, PEND)$. For each $X \in S$, T_j reads the version of X written by T_k such that T_k is the last transaction to write X before T_j in the topological order. After version assignment, the corresponding read-from arc (T_k, T_j) should be added to $SG_{mww}(<P, I>, q, PEND)$. Furthermore, an exclusion arc (T_j, T_l) of type (4) should also be added if $W_l[X]$ follows $W_k[X]$ in Pq without any intervening write operation on X.

Theorem 2.2: *$ATIO^{*}_{mww}(<P, I>, q, PEND)$ is acyclic if and only if $SG_{mww}(<P, I>, q, PEND)$ is acyclic.*

Proof. First note that all the read-from, constraint, and exclusion arcs introduced in $SG_{mww}(<P, I>, q, PEND)$ also exist in $ATIO^{*}_{mww}(<P, I>, q, PEND)$. Therefore, it is obvious that if $SG_{mww}(<P, I>, q, PEND)$ has a cycle, so does $ATIO^{*}_{mww}(<P, I>, q, PEND)$.

Conversely, assume that $SG_{mww}(<P, I>, q, PEND)$ is acyclic. It is sufficient to show that $ATIO_{mww}(<P, I>, q, PEND)$ has a DITS, since, as shown in [7], $ATIO_{mww}(<P, I>, q, PEND)$ has a DITS if and only if $ATIO^{*}_{mww}(<P, I>, q, PEND)$ is acyclic.

We first show that all the read-from and constraint arcs in $ATIO_{mww}(<P, I>, q, PEND)$ exist, or are implied by transitivity, in $SG_{mww}(<P, I>, q, PEND)$. Each read-from arc in $ATIO_{mww}(<P, I>, q, PEND)$ exists as a read-from arc also in $SG_{mww}(<P, I>, q, PEND)$. Let us now consider the constraint arcs in $ATIO_{mww}(<P, I>, q, PEND)$. Let $W_i[X]$ and $W_j[X]$ be two write operations such that $W_i[X] < W_j[X]$. Then $ATIO_{mww}(<P, I>, q, PEND)$ has a ww-arc (T_i, T_j). We want to show that $SG_{mww}(<P, I>, q, PEND)$ has a path from T_i to T_j.

If both $W_i[X]$ and $W_j[X]$ belong to Pq, then $SG_{mww}(<P, I>, q, PEND)$ has a path consisting only of ww-arcs introduced by rule (1). On the other hand, if $W_i[X]$ is in Pq and $W_j[X]$ belongs to $PEND$, then $SG_{mww}(<P, I>, q, PEND)$ has a path from T_i to T_k, where $W_k[X]$ is

the last write operation on X in Pq, and it also has a ww-arc (T_k, T_j) introduced by rule (2).

In addition to ww-arcs, both $ATIO_{mvw}(<P, I>, q, PEND)$ and $SG_{mvw}(<P, I>, q, PEND)$ have rw-arcs that are exactly those introduced by rule (6) above.

The rest is the same as the proof of Theorem 2.1 if $SG_{ww}(Pq, PEND)$ and $ATIO_{mvw}(Pq, PEND)$ are replaced by $SG_{mvw}(<P, I>, q, PEND)$ and $ATIO_{ww}(<P, I>, q, PEND)$, respectively. □

3. Updating Rules for Serialization Graphs

As we mentioned earlier, it would be too time-consuming to construct the serialization graph every time a new request arrived. Therefore, we maintain the serialization graph and make necessary modifications as new requests arrive and as they are granted/rejected. We will present a set of rules for updating serialization graphs; these rules depend on the type of scheduler that we employ. If the result of updating has a cycle, part of the modifications made are undone. A set of rules under (A) below make permanent changes, and their effects persist after the corresponding request is not granted immediately, but is delayed for future granting, while the changes made by rule (B) below are undone if the corresponding request is not granted.

We start with a graph consisting only of the initial transaction node T_0. As before, let q be the current request and $PEND$ be the set of pending requests. RS_h and WS_h stand for the readset and writeset of transaction T_h, respectively. When a new transaction T_h arrives, it declares its readset RS_h and writeset WS_h. At this time, we perform certain steps according to rule (A) below, before we consider T_h's first request as q and apply rule (B).

(A) If the first request of a new transaction T_h arrives, add a node for T_h to the serialization graph. Then after adding $\{R_h[X] \mid X \in RS_h\}$ and $\{W_h[X] \mid X \in WS_h\}$ to $PEND$, introduce the following constraint and exclusion arcs, where $W_k[X]$ is the last write on X in Pq. Rule numbers cited refer to those in Section 1.

(i) $CS(WW)$

For each $X \in WS_h$, introduce a ww-arc (T_k, T_h) due to rule (2), and for each T_j such that T_j reads X from T_k, introduce an exclusion arc (T_j, T_h) due to rule (5).

For each $R_h[X], X \in RS_h$, introduce a wr-arc (T_k, T_h) due to rule (3).

(ii) *MCS (MWW)*

Introduce the first two types of arcs used in *CS (WW)* above (i.e., those due to rules (2) and (5)).

(B) The arcs listed below are introduced for q, regardless of whether q is the first request of a new transaction or not.

(i) *CS (WW)*

$q = R_j[S]$:

For each $X \in S$, introduce a read-from arc (T_k, T_j), where $W_k[X]$ is the last write operation on X in Pq, and an exclusion arc (T_j, T_h) for each $W_h[X] \in PEND$ due to rule (5).

$q = W_j[S]$:

For each $X \in S$, introduce a ww-arc (T_j, T_h) due to (2) for each $W_h[X] \in PEND$ and a wr-arc (T_j, T_h) for each $R_h[X] \in PEND$ due to rule (3). Delete each arc (T_k, T_h) introduced previously due to rule (2) or (3), where $W_k[X]$ is the last write on X in P.

(ii) *MCS (MWW)*

$q = R_j[S]$:

For each $X \in S$, introduce a wr-arc (T_j, T_h) due to (6) for each $W_h[X] \in PEND$.

$q = W_j[S]$:

For each $X \in S$, introduce a ww-arc (T_j, T_h) due to (2) for each $W_h[X] \in PEND$. Delete each arc (T_k, T_h) introduced previously due to rule (2), where $W_k[X]$ is the last write on X in P.

Remark: Arcs due to rules (1) and (4) are never explicitly introduced in the above updating rules, since they are introduced by rules (2) and (5).

Theorem 3.1: *Given a serialization graph that corresponds to the ATIO graph just before q arrives, the above updating rules correctly construct a new serialization graph representing the*

new state.

Proof. Clear from the definition of the serialization graphs. □

Note that some of the arcs introduced by the updating rules may be redundant. For example, in the case of $q = R_j[S]$ for $CS(WW)$, a read-from arc (T_k, T_j) is introduced by the updating rule, but this arc already exists as a wr-arc, when $W_k[X]$ becomes the last write operation on X in Pq or when a new T_j arrives.

4. The Completion Test for *MCS*(*MWRW*)

4.1 The completion test for *MCS*(*MWRW*).

The completion test for $MCS(MWRW)$ is carried out by testing for a cycle in ATIO graph $ATIO^{*}_{mwrw}(<P, I>, q, PEND)$, which is acyclic if and only if the *mwrw*-serialization graph, $SG_{mwrw}(<P, I>, q, PEND)$, defined below, is acyclic [7].

The node set of $SG_{mwrw}(<P, I>, q, PEND)$ consists of the non-dummy nodes of $ATIO(<P, I>, q, PEND)$, and the read-from arcs are simply copied from this ATIO graph. If $q = R_j[S]$, a read-from arc for each $X \in S$ is introduced exactly as in $SG_{mww}(<P, I>, q, PEND)$. In addition to these arcs, some additional constraint arcs (wrw-arcs and some ww-arcs) are introduced, as described in subsections 4.1.1 and 4.1.2 below. To facilitate our discussion, we first introduce some new terms. A write operation whose update is (not) read by some other transaction is called **useful write** (**useless write**). For each $X \in \mathbf{D}$, let S_X denote the subsequence of Pq consisting of all the read and write operations on X. S_X can be considered as a sequence of runs, where each run consists of a maximal sequence of consecutive read (or write) operations. A run of write operations having a useful write in it is called a **useful run.**

In the single-version case, all runs of writes, except possibly for the last one, are clearly useful, because they are followed by a run of reads. It turns out that this is also true in the multiversion case as stated in the following lemma. However, unlike in the single-version case, the

useful write is not necessarily the last write in a useful run.

Lemma 4.1: *For each $X \in \mathbf{D}$, there is exactly one useful write operation in each run of writes on X in P, except for the last run. The last run of writes has at most one useful write operation.*

Proof. Let R stand for any run of writes in P, and suppose there are two useful writes $W_i[X]$ and $W_h[X]$ in R. Let T_j and T_l read from T_i and T_h, respectively, so that (T_i, T_j) and (T_h, T_l) are read-from arcs. Since these writes are in the same run, they precede both of the read operations, $R_j[X]$ and $R_l[X]$, giving rise to wr-arcs (T_i, T_l) and (T_h, T_j). Then an exclusion arc (T_j, T_h) will be induced due to the wr-arc (T_i, T_j). This exclusion arc and the wr-arc (T_h, T_j) form a cycle in $ATIO^*_{mwrw}(<P, I>, q, PEND)$, which contradicts the correctness of $MCS(MWRW)$ proved in [7]. Thus $MCS(MWRW)$ would not have made these version assignments to $R_j[X]$ or $R_l[X]$.

Now let R stand for a run of writes other than the last one. If there is no useful write in R, then the version written by a write $W_i[X]$ in a run preceding R is assigned to the read operation $R_j[X]$ in P that immediately follows R. Let $R_l[X]$ be a read operation between $W_i[X]$ and R in P and let $W_g[X]$ be in R. In $ATIO^*_{mwrw}(<P, I>, q, PEND)$, there is a path from T_i to T_g due to wr-arc (T_i, T_l) and rw-arc (T_l, T_g). There is also an exclusion arc (T_g, T_i) due to the two intervals labeled X originating from T_i and T_g, respectively, and wr-arc (T_g, T_j). Thus, $ATIO^*_{mwrw}(<P, I>, q, PEND)$ has a cycle, which again contradicts the correctness of $MCS(MWRW)$. □

4.1.1 Constraint arcs due to the operations in Pq.

We scan S_X from the beginning to the end. Let p be the current operation under scan.

Case 1: $p = R_i[X]$.

If the current run (i.e., the run which contains p) is followed by a run R' of writes, then introduce the following rw-arcs.

(7) rw-arc (T_i, T_j) for each useless write $W_j[X]$, if any, in R'.

If R' consists of a single useful write $W_l[X]$ and no useless write, then introduce

(8) rw-arc (T_i, T_l).

Remarks:

(a) Not all rw-arcs are used, since others are implied by read-from arcs and ww-arcs introduced below.

(b) Rule (7) doesn't apply if $W_j[X]$ is useful, since such a constraint will be implied from rule (9) below.

Case 2: $p = W_i[X]$.

If the current run of writes R is useful and p is useless, then introduce

(9) ww-arc (T_i, T_j), where $W_j[X]$ is the useful write in R.

Remark:

A ww-arc due to rule (9) is not an additional constraint, but actually is implied by a dummy arc $(T_i, T_i'):X$ and an exclusion arc, which exists in $ATIO^*_{mwrw}(<P, I>, q, PEND)$ due to the two intervals labeled by X originating from T_i and T_j, respectively, and the wr-arc from T_i to the transaction node that reads X from T_j.

4.1.2 **Constraint arcs due to pending operations on each $X \in$ D.**

Let L' and L be the second last and the last runs in S_X, respectively.

Case 1: L' is a run of writes and L is a run of reads.

Introduce

(10) rw-arc (T_i, T_k) for each $R_i[X]$ in L and for each $W_k[X] \in PEND$, and

(11) wr-arc (T_k, T_k) for each $R_k[X] \in PEND$, where $W_k[X]$ is the last write operation X in L'.

Case 2: L' is a run of reads and L is a run of writes.

Introduce

(12) rw-arc (T_i, T_k) for each $R_i[X]$ in L' and for each $W_k[X] \in PEND$.

(13) wr-arc (T_i, T_k) for each $W_i[X]$ in L and for each $R_k[X] \in PEND$.

The *mwrw*-serialization graph $SG_{mwrw}(<P, I>, q, PEND)$ has, in addition to the read-from arcs, constraint arcs (7) to (13) defined above.

4.2 Updating Rules

When the first request of a new transaction arrives, we first perform certain steps according to rule (A) below, before we consider this request as q and apply rule

(A) If the first request of a new transaction T_h arrives, add a node for T_h to the serialization graph. Then add $\{R_h[X] \mid X \in RS_h\}$ and $\{W_h[X] \mid X \in WS_h\}$ to $PEND$. We state below constraint and exclusion arcs to be added for $MCS(MWRW)$. Rule numbers cited refer to those in Section 4.1.

For each $R_h[X], X \in RS_h$, introduce wr-arcs due to rules (11) and (13).

For each $W_h[X], X \in WS_h$, introduce rw-arcs due to rule (10) and (12).

(B) The changes specified below are made for q, regardless of whether q is the first request of a new transaction or not. The changes made are undone if q is not granted.

$q = R_j[S]$:

Delete the wr-arcs of the form (T_i, T_j) introduced earlier by rule (11) or (13). If S_X ends in a write step for some $X \in S$ (i.e., there is a new run consisting only of $R_j[X]$ at the end of S_X), then delete all arcs previously introduced due to rule (12). Finally introduce rw-arcs (T_j, T_h) due to (10). Note that the version assignments to this read step are carried out *after* the completion test (see the remarks below).

$q = W_j[S]$:

Delete the rw-arcs of the form (T_i, T_j) introduced earlier by rule (10) or (12). If S_X ends in a read step for some $X \in S$ (i.e., there is a new run consisting only of $W_j[X]$ at the end of S_X), then delete all arcs previously introduced due to rule (11). Finally introduce wr-arcs (T_j, T_h) due to (13).

Remarks:

A multiversion cautious scheduler such as $MCS(MWRW)$ makes the version assignment *after* each completion test. Version assignments introduce read-from arcs in the serialization graph, which changes the useful/useless status of a write operation. Therefore, after each version assignment, an arc introduced earlier by rule (7) may have to be deleted, and new arcs due to rules (8) and (9) may have to be introduced.

5. How to Reduce Serialization Graph

In Sections 3 and 4, we discussed how to update a serialization graph to reflect a new request q. If the resulting serialization graph has a cycle, q cannot be granted, and the old serialization graph without the effect of q must be restored. As time progresses, the number of nodes and arcs in a serialization graph becomes larger and larger, and finding a cycle in it will incur more and more overhead. Therefore, we should keep the size of the graph as small as possible by eliminating those nodes and arcs which do not affect future scheduling decisions. We now define the erasable transaction recursively.

Definition 5.1: A source node (and the corresponding transaction) of a serialization graph that has completed its steps is said to be **erasable**. A node (and the corresponding transaction), which has completed its steps, is erasable if all its ancestors in the serialization graph are erasable. A node that is not erasable is said to be **nonerasable**. □

It is easily seen that T_0 is erasable. Note that a completed transaction is not always erasable, since all its ancestors may not have completed their steps.

We want to show that an erasable node can never become a part of a cycle, and therefore can be eliminated from the serialization graph without destroying any cycle.

Lemma 5.1: *Under the updating rules of Sections 3 and 4, except for rule (9), an erasable node will never acquire an incoming arc.*

Proof. The updating rules in Sections 3 and 4, except for rule (9), never introduce an arc *to* a completed transaction. □

Since rule (9) is never used by *CS*(*WW*) or *MCS*(*MWW*), it follows from Lemma 5.1 that the erasable nodes can be eliminated in these schedulers.

We now deal with *MCS*(*MWRW*), since it may invoke rule (9). We wish to be able to delete erasable nodes in this case as well. To this end, we introduce the following rule for topological sorting.

(*) *In finding a topological sort for the SG, if there is more than one source node, order erasable source nodes before nonerasable ones.*

With rule (*), when rule (9) is invoked after a write operation $W_j[X]$ is assigned to the current

read operation, $R_i[X]$, if T_j is erasable then so is T_k, where ww-arc (T_k, T_j) is introduced by rule (9). This implies that no arc is introduced from a nonerasable node to an erasable one. In addition, arcs due to rule (9) alone do not create a cycle in the SG. For an arc is introduced by rule (9) only when a version assignment is made, and is consistent with a topological sort of the SG. It follows that any erasable node can be eliminated from the SG graph.

Let us now discuss how to delete unnecessary transactions efficiently. When the last step of transaction T_i has been granted, T_i and possibly some other transactions may become erasable. We repeat the following operation as many times as possible:

- *Find a source node (transaction) in the serialization graph that has completed its steps and delete it together with its outgoing arcs.*

If the above operation is no longer applicable, the serialization graph is said to be **reduced**. The updating rules should be so modified that a new arc is generated only if its end nodes have not been erased, i.e., they are still present in the reduced serialization graph.

A data item in the readset or writeset of a nonerasable transaction is said to be **active**. We only keep track of nonerasable transactions and active data items.

6. Testing Acyclicity of Serialization Graph

We use a standard method for testing acyclicity of a graph, namely successive source marking; if all nodes are marked, the graph is acyclic, otherwise it contains a cycle. Suppose that the serialization graph SG under consideration is reduced. We introduce the following data structures for SG.

NERSBL = set of all nonerasable transactions, i.e., the node set of SG,

$A_{SG}(T_i)$ = adjacency list for node T_i, i.e., a list containing all T_j such that (T_i, T_j) is an arc in SG.

$COUNT_{SG}(T_i)$ = the indegree of node T_i, i.e., the number of arcs incoming to node T_i of SG.

We present below the acyclicity test as a formal procedure. It should be clear that the procedure works correctly.

Procedure ACYCLIC

Input: $A_{SG}(T_i)$, $COUNT_{SG}(T_i)$ and $NERSBL$ for serialization graph SG.

Output: DITS order of SG (as an array p such that $p(T_i)$ represents T_i's position in the DITS), if it is acyclic; a message "SG has a cycle", otherwise.

Step 0: Set $NSET := NERSBL$, $n := 0$.

Step 1: **while** there is a $T_i \in NSET$ such that $COUNT_{SG}(T_i)=0$ **do:**

Set $n := n+1$, $p(T_i) := n$, $NSET := NSET - \{T_i\}$, and for each $T_j \in A_{SG}(T_i)$, $COUNT_{SG}(T_j) := COUNT_{SG}(T_j) - 1$.

Step 2:

if $n = |NERSBL|$ **then** output "SG is acyclic" (p represents a DITS order).

else output "SG has a cycle" and halt. □

The above procedure is applied after the updating corresponding to a new request q has been carried out. If a cycle is detected, then q is delayed and the updating must be undone. For this purpose, the new nodes and arcs that result from updating should be kept in a temporary storage, so that the old SG can be restored quickly. If q is granted, the temporary changes should be made permanent.

7. Committing Transactions

Although the cautious scheduler never aborts a transaction for the purpose of concurrency control, a transaction may voluntarily abort itself before it completes all its steps, due to errors or for some other reasons [4]. If a transaction aborts itself, then all the transactions that have read some data item(s) from it must be rolled back. A completed transaction is said to have been **committable** if it is never necessary to roll back the transaction, due to the abortion of some other uncompleted transaction.

To determine whether a transaction is committable, we define the **commit graph** (CG, for short), which is a subgraph of the serialization graph; in general, the node set and arc set of the CG are subsets of those of the corresponding SG. Its node set consists of all non-committed

transactions, and its arc set consists only of the read-from arcs among the nodes. With the commit graph a necessary and sufficient condition for committing a transaction T_i can be stated as follows.

Commit Condition: *T_i has completed all its steps and so have all transactions that are ancestors of T_i in CG.*

Note that an erasable transaction in *SG* is committable, but not vice versa. For example, consider two transactions $T_1 = R_1[X]W_1[Y]$ and $T_2 = R_2[X]W_2[X]$. Suppose *MCS(MWRW)* has reached a point where $P = R_1[X]R_2[X]W_2[X]$ and $PEND = \{W_1[Y]\}$. At this point, T_2 is committable, but nonerasable, because of an rw-arc (T_1, T_2).

To maintain CG, in addition to SG, of course incurs overhead. We may choose to maintain only SG and commit only erasable transactions. This way, the committing of some committable transactions may be delayed, but we can do without CG.

8. Complexity Analysis

We introduce the following notations associated with a reduced serialization graph SG. Recall that such a graph has only nonerasable nodes.

$T_{pr}(X) = \{\text{transaction } T_i \text{ in SG} \mid R_i[X] \in PEND\}$

$T_{pw}(X) = \{T_i \text{ in SG} \mid W_i[X] \in PEND\}$

$T_{cr}(X) = \{T_i \text{ in SG} \mid R_i[X] \text{ has been completed}\}$

$T_{cw}(X) = \{T_i \text{ in SG} \mid W_i[X] \text{ has been completed}\}$

$T_{cr}(X)$ and $T_{cw}(X)$ can be replaced by their subsets, depending on the particular scheduler under consideration as follows.

CS(WW):

$T_{cw}(X) = \{T_i \text{ in SG} \mid T_i \text{ that has written } X \text{ most recently}\}$

$T_{cr}(X) = \{T_i \text{ in SG} \mid T_i \text{ that has read from the transaction in } T_{cw}(X)\}$

For $MCS(MWW)$ and $MCS(MWRW)$, we represent $T_{cw}(X)$ as a doubly linked list, arranged in the order that the write operations are completed, so that the last write operation on X in it can be found in constant time.

Consider a certain time instant τ during the course of cautious scheduling and define

t_{SG}: the number of nodes in SG

t_{CG}: the number of nodes in CG

a_{SG}: the number of arcs in SG

a_{CG}: the number of arcs in CG.

All these are defined with respect to τ. Since the commit graph CG is always a subgraph of the serialization graph SG, we have $t_{CG} \le t_{SG}$ and $a_{CG} \le a_{SG}$. Assuming that a new request q by transaction T_i has just arrived, we analyze the time complexity required for processing q.

Consider the case where T_h is a new transaction. Updating rule (A) introduces at most the following number of arcs.

$$\sum_{X \in RS_h} |T_{cw}(X)| + \sum_{X \in WS_h} (|T_{cw}(X)| + |T_{cr}(X)|). \tag{16}$$

The first sum above accounts for the wr-arcs introduced by rule (A). The second sum accounts for the ww-arcs and exclusion arcs (for $CS(WW)$ and $MCS(MWW)$), and the rw-arcs (for $MCS(MWRW)$), all introduced by rule (A). Updating data structures cost constant time per arc by using $T_{cw}(X)$ and $T_{cr}(X)$. Define $Size(T_h)$ by

$$Size(T_h) = |RS_h| + |WS_h|.$$

Since $|T_{cw}(X)| \le t_{SG}$ and $|T_{cr}(X)| \le t_{SG}$, the time required for executing updating rule (A) is thus

$$O(Size(T_h) \cdot t_{SG}) \tag{17}$$

Let $|q|$ denote the number of operations in step q. Since the number of arcs introduced by updating rule (B) is at most

$$\sum_{R_j[X] \text{ or } W_j[X] \in q} (|T_{pw}(X)| + |T_{pr}(X)|),$$

excluding those due to rule (14). Since $|T_{pw}(X)| \leq t_{SG}$ and $|T_{pr}(X)| \leq t_{SG}$ hold, and the application of rule (9) doesn't involve more than t_{SG} arcs, we obtain the time required for executing updating rule (B), including deletions:

$$O(|q| \cdot t_{SG}). \tag{18}$$

As indicated after procedure ACYCLIC in Section 5, we must copy the original $COUNT_{SG}(T_i)$ and $A_{SG}(T_i)$ before updating the serialization graph SG. This work requires time

$$O(t_{SG} + a_{SG}). \tag{19}$$

We now analyze the acyclicity test of the serialization graph SG. It is clear that ACYCLIC runs in time

$$O(|NERSBL|) + \sum_{T_i \in NERSBL} O(|COUNT_{SG}(T_i)| + |A_{SG}(T_i)|).$$

Let t'_{SG} and a'_{SG} denote the numbers of nodes and arcs, respectively, in the updated serialization graph. Since

$$|NERSBL| = t'_{SG}$$

and

$$\sum_{T_i \in NERSBL} |COUNT_{SG}(T_i)| = \sum_{T_i \in NERSBL} |A_{SG}(T_i)| \leq a'_{SG},$$

the running time of ACYCLIC is

$$O(t'_{SG} + a'_{SG}). \tag{20}$$

If q is delayed, the serialization graph is rolled back to the state just before the addition of arcs by rule (B), and nothing else need to be done. Therefore, we consider the case where q is granted in what follows. In this case, we need to do the following.

(a) Delete all the erasable transactions from the serialization graph SG.

(b) Commit all the committable transactions, if any.

(c) Make version assignments to $q = R_i[S]$ in the case of a multiversion scheduler.

To carry out (a), we need to identify the erasable transactions. They can be found by using the indegree of each node in the SG, which is available in $COUNT_{SG}(T_i)$ used in the procedure ACYCLIC. Actual deletion doesn't take more than

$$O(t_{SG} + a_{SG}) \tag{21}$$

time. To perform (b), note that if the SG is acyclic then so is the CG, and a topological sort of the SG is a topological sort for the commit graph CG, if it is projected on to the node set of the CG. A topological sort of the SG is obtained as a by-product of invoking procedure ACYCLIC in the completion test, hence, at no extra cost. Actual committing can be done in

$$O(t_{CG} + a_{CG}) \tag{22}$$

time, where t_{CG} and a_{CG} denote the numbers of nodes and arcs, respectively, in the CG. As for (c), the version assignment for each $R_i[X]$, $X \in S$, can be done in $O(t_{SG})$ time. See the version assignment rules in [7] for the multiversion case. Thus a total of

$$O(|q| \cdot t_{SG}) \tag{23}$$

time is required.

We now summarize the above results in Table 8.1, where

N_{tot} = Total number of transactions

N_{ct} = Total number of completion tests

N_{rs} = Total number of read steps.

When we estimate the time requirements of cautious schedulers, we need to use the average values of t_{SG}, a_{SG}, t_{CG} and a_{CG} over the time during which the scheduler is used. Let $\bar{t}_{SG}$, $\bar{a}_{SG}$, $\bar{t}_{CG}$ and $\bar{a}_{CG}$ denote their average values, respectively. Recall that we have $t_{CG} \le t_{SG}$ and $a_{CG} \le a_{SG}$. Also it is clear that $N_{tot} \le N_{ct}$ (if each transaction has a read step) and $N_{rs} \le N_{ct}$ hold. From Table 8.1, we obtain the total time required:

	Time complexity per application	No. of applications
Updating rule (A)	$O(Size(T_h)\cdot t_{SG})$ from (17)	N_{tot}
Updating rule (B)	$O(\mid q \mid \cdot t_{SG})$ from (18)	N_{ct}
Acyclicity test	$O(t_{SG} + a_{SG})$ from (19) & (20)	N_{ct}
Erasing transaction	$O(t_{SG} + a_{SG})$ from (21)	N_{tot}
Committing transaction	$O(t_{CG} + a_{CG})$ from (22)	N_{tot}
Version assignment	$O(\mid q \mid \cdot t_{SG})$ from (23)	N_{rs}

Table 8.1: Breakdown of complexity.

$$O(N_{tot}\cdot\overline{Size}\cdot\bar{t}_{SG}) + O(N_{ct}(\overline{\mid q \mid}\cdot\bar{t}_{SG} + \bar{t}_{SG} + \bar{a}_{SG})) + O(N_{tot}(\bar{t}_{SG} + \bar{a}_{SG})),$$

where $\overline{Size}$ and $\overline{\mid q \mid}$ are the averages of $Size(T_h)$ and $\mid q \mid$, respectively. The first term accounts for the time for updating the SG using rule (A), and the second term accounts for the completion test. As for the third term, it represents the time needed for assigning versions, reducing the SG graph, and committing transactions. The above expression will in most cases reduce to

$$O(N_{ct}\cdot\overline{Size}\cdot\bar{t}_{SG}).$$

References

1. G. N. Buckley and A. Silberschatz, Obtaining progressive protocols for a simple multiversion database model, *Proc. 9th Int. Conf. on VLDB*, , Oct/Nov 1983, 74-80.

2. M. A. Casanova and P. A. Bernstein, General purpose schedulers for database systems, *Acta Informatica 14*, (1980), 195-220.

3. M. A. Casanova, The concurrency control problem for database systems, in *Lecture Notes in Computer Science 116*, Springer Verlag, Berlin, 1981.

4. T. Ibaraki and T. Kameda, Cautious schedulers under incomplete predeclarations (in Japanese), Proc. 3rd Data Engineering Seminar,, IEICE of Japan, Nov. 1987.

5. T. Ibaraki, T. Kameda and T. Minoura, Serializability with constraints, *ACM Trans. Database Systems 12*, 3 (Sept. 1987), 429-452.

6. T. Ibaraki, T. Kameda and N. Katoh, Cautious transaction schedulers for database concurrency control, *IEEE Trans. SE 14*, 7 (July 1988), 997-1009.

7. T. Ibaraki, T. Kameda and N. Katoh, Multiversion cautious transaction schedulers for database concurrency control, *IEEE Trans. SE 16*, 3 (March 1990), 302-315.

8. N. Katoh, T. Ibaraki and T. Kameda, Cautious transaction schedulers with admission control, *ACM Trans. Database Systems 10*, 2 (June 1985), 205-229.

9. S. Nishio, S. Taniguchi and T. Ibaraki, On the efficiency of cautious schedulers for database concurrency control - Why insist on two-phase locking?, *Jour. of Real-Time Systems 1*, (1989), 177-195.

Centre for Systems Science

• Simon Fraser University established the **Centre for Systems Science** as a provincial centre for advanced study. The objective of CSS is twofold: coordinate and combine advanced systems research at SFU, and encourage application of research results within industry in order to further the technological development of British Columbia.

• CSS has assembled three research groups to streamline its efforts. The orientation of these groups is as follows: intelligent systems, microelectronics, and computer and communications systems. Because of shared research interests, this latter group merges with LCCR (**Laboratory for Computer and Communications Research**) for the publication of technical reports.

• CSS's three groups involve over 80 skilled faculty researchers principally from SFU's Schools of Engineering Science and Computing Science, but also draw substantially on Mathematics, Physics, Geography, Philosophy, Linguistics, Psychology, Communication, and Kinesiology.